Express Mail Mailing Label No. EL 988705041 US

PATENT APPLICATION
Atty. Docket No.: SNS-015 (7268/23)

APPARATUS AND METHODS FOR TEXTURE MAPPING

Related Applications

[0001] This application is related to the commonly-owned U.S. patent application entitled, "Apparatus and Methods for Stenciling an Image," by Levene et al., filed under Attorney Docket No. SNS-016 on even date herewith, the text of which is hereby incorporated by reference in its entirety.

Field of the Invention

[0002] This invention relates generally to graphical rendering. More particularly, in certain embodiments, the invention relates to texture mapping techniques for the graphical rendering of a virtual object.

Background of the Invention

[0003] Certain computer graphics applications graphically render the surface of a three-dimensional virtual object. The surface of a three-dimensional virtual object is generally represented as a mesh of polygonal surface elements, for example, triangles. Attributes, such as color intensity, are assigned to surface elements of the virtual object, which are then displayed to the user.

[0004] The resolution of the surface mesh is typically chosen according to the three-dimensional complexity of the object. For example, rendering a virtual object having a complex geometry may require a tight mesh of many small surface elements, whereas rendering a virtual object of simpler shape may be performed with a looser mesh of fewer, larger surface elements.

[0005] It is typically not possible to achieve sufficient realism by assigning color values to vertices of each surface element, particularly if the surface elements are large. This problem may be overcome by representing three-dimensional surface properties of the object using a texture in two-dimensional space. Texture mapping permits improved resolution of surface properties, such as color values, within each surface element, and allows certain operations to be performed on the three-dimensional object as if it were flat.

[0006] Mapping methods are used to relate points on the surface of a three-dimensional virtual object to corresponding points of a two-dimensional surface. For example, mapping methods can be used to relate points on the surface of a three-dimensional sphere, such as the Earth, to corresponding points of a two-dimensional map. Certain graphics applications perform texture mapping for a three-dimensional virtual object by establishing a global parameterization that links every surface element in the three-dimensional object space to an element in two-dimensional texture space. The mapping process may not be automatic and may require manual input from the user. A goal of these mapping schemes is to produce a coherent two-dimensional texture whose elements are arranged in such a way that the distortion of the object surface is acceptably low. For many complex virtual objects, this poses a problem that is either computationally difficult or intractable.

[0007] Furthermore, current graphics applications require a user to paint the surface of a virtual object in two-dimensional texture space before applying the texture on the virtual object in three-dimensional object space. This results in a less interactive, less intuitive experience for the user.

[0008] Moreover, current methods do not allow a user to modify the shape of the object after its surface has been painted without losing surface data from unmodified portions of the object. This is due, in part, to the need to re-mesh and re-parameterize the entire model following any modification of the underlying three-dimensional model.

[0009] A method of texture painting has recently been introduced to allow texture mapping without global parameterization. M. Foskey et al., "ArtNova: Touch-Enabled 3D Model Design," *IEEE Virtual Reality 2002*, pp. 119-126, (March 2002).

[0010] However, there remains a need for mapping methods that allow a user to modify the shape of an object after its surface has been painted, without losing the surface data from unmodified portions of the object. There also exists a need for a more efficient graphical rendering method that is able to support the rendering of complex virtual objects, while still allowing a user to interactively paint directly onto the object in object space. Furthermore, there is a need for a method of blending textures while maintaining the interactivity of painting, without creating graphical artifacts.

Summary of the Invention

[0011] The invention provides improved methods of graphically rendering virtual objects. More specifically, the invention provides improved texture allocation and texture rendering techniques for graphically representing the surface of a three-dimensional virtual object.

[0012] The improved methods allow a user to paint directly onto the surface of a three-dimensional virtual object. The methods also allow a user to interactively modify the shape of a painted, three-dimensional object without losing surface data from unmodified portions of the model. For example, the improved methods permit a user to paint an object, then carve portions of the object without losing the painted surface data from unmodified portions of the model. The improved methods also permit a user to easily switch back and forth between a painting mode and a sculpting mode, without having to reallocate texture for unmodified portions of the object.

[0013] Additionally, the invention provides techniques for blending a brush stroke into a surface texture without creating artifacts in the overlapping segments of the stroke, and without diminishing the interactivity of the user's painting experience.

[0014] Furthermore, the invention features improved methods for the combined graphical and haptic rendering of a virtual object, thereby permitting a user to experience force feedback via a haptic interface device during the painting of the object in object space. In this way, a user can "feel" the virtual object as the user paints its surface while, at the same time, the user observes a display of the object in object space.

[0015] Texture mapping methods of the invention include texture allocation methods and texture rendering methods. Texture allocation refers to the process of allocating elements in texture space to corresponding surface elements in object space. Texture rendering refers to the process by which data in the texture space is sent to a graphics card and/or graphics application for display. Texture rendering also involves sending information to the graphics application that allows the application to correctly relate the texture data to corresponding portions of the surface of the virtual object. Texture rendering methods of the invention are able to interface with any commercially-available standard for three-dimensional rendering and/or three-dimensional hardware acceleration, including cross-platform standards.

[0016] In one embodiment, the invention provides an improved method for graphically rendering a virtual object by using information produced during mesh generation. In the process of generating a surface mesh for a voxel-based virtual object, an index is produced for each resulting jack, where a jack is a surface-containing voxel, and a voxel is a volumetric element in object space. Each jack contains a certain configuration of surface elements. There are a finite number of possible configurations of surface elements within each jack; and the number of possible configurations depends on the mesh generation scheme employed. The index for each jack corresponds to one of the finite number of surface element configurations. A first lookup table is created by determining a local configuration of texture elements for each of the known set of surface element configurations. The first lookup table is preferably computed prior to object rendering. Thus, graphical rendering of a given virtual object proceeds by using the indices for the jacks of the virtual object to generate coordinates of texture elements in texture space corresponding to the surface elements of the virtual object.

[0017] Thus, in one embodiment, the invention provides a method of graphically rendering a virtual object including the steps of: (1) using an index corresponding to each of a plurality of jacks of a voxel-based virtual object to identify texture elements to which surface elements of the virtual object are mapped; and (2) generating texture coordinates in texture space for each of the identified texture elements. The index may be, for example, a marching cubes index, derived from an implementation of a marching cubes algorithm that is used to generate a surface mesh for the virtual object. Alternatively, the index may be, for example, a marching tetrahedra index, derived from an implementation of a marching tetrahedra algorithm used to generate a surface mesh for the virtual object.

[0018] It is found that dividing the surface of a virtual object into texture regions may overcome hardware limitations in rendering complex virtual objects. Thus, in one embodiment, the texture space in the graphical rendering method summarized above includes a plurality of texture regions. Moreover, in addition to steps (1) and (2) above, the rendering method may further include the steps of: (3) binding to a graphics application a blended texture corresponding to one of the plurality of texture regions; and (4) transmitting to the graphics application the texture coordinates for each of a plurality of surface elements associated with the blended texture. The blended texture may be a contiguous texture formed by blending together a plurality of texture layers, as discussed in more detail hereinbelow.

[0019] The index produced for each jack as a result of the mesh generation procedure can be used to create any number of lookup tables. In one embodiment, the graphical rendering method includes creating two lookup tables, using a first lookup table to determine to which of a plurality of texture elements a given surface element is mapped, and/or using a second lookup table to determine to which of a plurality of positions within a texture element a given surface element is mapped. For example, where either zero, one, or two triangular surface elements in object space are mapped to a quadrilateral texture element in texture space, a first lookup table is used to determine to which quadrilateral a given triangle is mapped, and the second lookup table is used to determine to which of two positions within the quadrilateral the triangle is mapped. More lookup tables may be used in more complex embodiments, for example, where all of the surface elements of a given jack are mapped to a single texture element.

[0020] The invention also includes methods of dynamic texture allocation and deallocation, allowing a user to interactively modify the shape of a painted, three-dimensional model, without losing texture data associated with unmodified portions of the object. Thus, the invention provides a method of mapping texture onto a virtual object that includes: allocating texture for at least one newly-created jack of a virtual object following an object modification; and/or de-allocating texture in the texture space for at least one newly-eliminated jack of the virtual object following an object modification.

[0021] In one embodiment, the method proceeds by stepping through contiguous blocks of jacks of the model in render order to determine whether texture allocation is needed therein, and by keeping texture corresponding to a given jack block together within the same texture region. This dynamic allocation method provides improved efficiency, since new texture is only created for the modified jacks, and since there is no global parameterization required. Advantages are achieved by allocating texture in render order and by keeping texture corresponding to a given jack block together within the same texture region. This promotes efficient binding and results in a more efficient texture rendering process. In one embodiment, texture is allocated so as to minimize the number of texture binds required during rendering.

[0022] The invention also provides for the compact storage of texture information for a virtual object. Since many texture elements may contain a uniform value, it is efficient to store this information as a single value, instead of storing values for all texels of every texture element. It is not necessary to store a full contiguous texture for a given texture region, since the user does not need to see the texture space at any time, as all edits may be performed by the user in object space. Accordingly, the invention provides a method of creating a blended texture for use in the rendering of a virtual object, the method including the step of blending two or more texture layers, where at least one of the texture layers is a grid of pointers indicating at least two of: (i) uniform texture elements, (ii) nonuniform texture elements, and (iii) the location of the nearest free texture element in the grid.

[0023] A blended texture may be created by combining scratch textures, stencil textures, and/or paint textures associated with one or more brush strokes. A scratch texture indicates one or more intensity values of pixels associated with each brush stroke, and a paint texture indicates one or more color values associated with each brush stroke.

[0024] The methods of blending texture presented herein are compatible with a method of stenciling in which a user may specify a maximum amount by which selected portions of an image are allowed to change over the course of a series of brush strokes. A first type of stencil texture indicates a level of protection to apply to a given region of an image, and a second type of stencil texture contains brush strokes that are accumulated and are modified according to the protection levels of the first type of stencil texture prior to blending into the image.

[0025] Methods of the invention include blending texture layers of at least two of the three types listed above – scratch texture layers, paint texture layers, and stencil texture layers – in the order in which the brush strokes are performed. Enhanced interactivity is achieved, since texture data is blended into the image pixel-by-pixel in discrete time slices, without interrupting the user's application of brush strokes. Additional model complexity can be handled, since texture layers are blended into composite textures for only those texture regions affected by the brush strokes. Furthermore, one embodiment stores the texture layers in a "collapsed" form that includes both uniform and nonuniform texture elements, and then combines them to form a contiguous blended texture suitable for binding to a graphics application.

[0026] By combining the various texture allocation, texture rendering, and texture blending methods disclosed herein, the invention provides a complete graphics application feature that allows a user to paint directly on the surface of a virtual object in object space. Accordingly, the invention provides a method of interactively representing application by a user of at least one brush stroke directly onto a virtual object in object space, the method comprising the steps of: (a) allocating a plurality of texture elements in two-dimensional texture space for a plurality of jacks of a virtual object; (b) graphically rendering the allocated texture in real time as a user applies at least one brush stroke onto the virtual object, wherein rendering includes creating at least one blended texture for binding to a graphics application; and (c) updating the blended textures according to the at least one brush stroke applied by the user, wherein the method involves one or more of the following: (i) using an index corresponding to each of a plurality of jacks of the virtual object to identify texture elements to which surface elements of the virtual object are mapped; (ii) dynamically allocating texture in the texture space; and (iii) blending a set of texture layers corresponding to a first of a plurality of texture regions in the texture space and binding the blended texture to the graphics application during rendering of the first texture region.

[0027] Finally, the invention features an architecture for combined graphical rendering and haptic rendering of a virtual object, allowing a user to experience force feedback during the painting of the object in object space.

Brief Description of the Drawings

[0028] The objects and features of the invention can be better understood with reference to the drawings described below, and the claims. The drawings are not necessarily to scale, emphasis instead generally being placed upon illustrating the principles of the invention. In the drawings, like numerals are used to indicate like parts throughout the various views. The patent or application file contains at least one drawing executed in color. Copies of this patent or patent application publication with color drawing(s) will be provided by the U.S. Patent and Trademark Office upon request and payment of the necessary fee.

[0029] Figure 1A is schematic diagram illustrating a surface mesh of a three dimensional virtual object, according to an illustrative embodiment of the invention.

[0030] Figure 1B is a schematic diagram illustrating a flat, coherent texture map of the object of Figure 1A, produced using global UV parameterization.

[0031] Figure 2A is a schematic diagram illustrating a three dimensional block with a painted face as viewed in object space, according to an illustrative embodiment of the invention.

[0032] Figure 2B is a schematic diagram illustrating texture in a two-dimensional texture space that is mapped to the block of Figure 2A, according to an illustrative embodiment of the invention.

[0033] Figure 3 is a block diagram featuring a method of texture allocation, according to an illustrative embodiment of the invention.

[0034] Figure 4A is a schematic diagram illustrating the partitioning of texture elements associated with a three dimensional virtual object into texture regions in texture space, according to an illustrative embodiment of the invention.

[0035] Figure 4B is a schematic diagram illustrating a volume in object space containing jack blocks, each made up of 512 contiguous voxels, at least one of which is a jack (a surface-containing voxel), where the volume contains surface elements that are mapped to a given texture region in texture space, according to an illustrative embodiment of the invention.

[0036] Figure 5 is a schematic diagram illustrating a hash map of hash maps identifying each texture element within a given texture region, where each texture element is mapped to a jack within a jack block, according to an illustrative embodiment of the invention.

[0037] Figure 6 is a schematic diagram featuring a method of texture rendering that includes binding a blended composite texture for a given texture region to a graphics application and sending 2D texture coordinates to a graphics card, according to an illustrative embodiment of the invention.

[0038] Figure 7 is a block diagram featuring a method of texture rendering, according to an illustrative embodiment of the invention.

[0039] Figure 8A is a schematic diagram illustrating a method of texture rendering using two look-up tables, according to an illustrative embodiment of the invention.

[0040] Figure 8B is a block diagram featuring the texture rendering method illustrated in Figure 8A applied to each surface element within each jack, according to an illustrative embodiment of the invention.

[0041] Figure 9 is a block diagram featuring a method of creating the first lookup table used in the texture rendering method of Figures 8A and 8B, according to an illustrative embodiment of the invention.

[0042] Figure 10 is a schematic diagram illustrating a method of identifying the texture elements associated with each surface element of a jack and generating texture coordinates for the identified texture elements, using the pre-computed lookup tables featured in Figure 8A, according to an illustrative embodiment of the invention.

[0043] Figure 11 is a schematic diagram illustrating a method of blending texture layers to form a composite blended texture, according to an illustrative embodiment of the invention.

[0044] Figure 12 is a schematic diagram illustrating a method of representing a texture layer as a grid of pointers, according to an illustrative embodiment of the invention.

[0045] Figure 13 is a schematic diagram illustrating two pillbox segments of a brush stroke both with and without a double-blending artifact in the overlapping portion, according to an illustrative embodiment of the invention.

[0046] Figure 14A is a schematic diagram illustrating the partitioning of the three dimensional virtual object into texture regions, according to an illustrative embodiment of the invention.

[0047] Figure 14B is a schematic diagram illustrating a method of blending multiple texture layers for each texture region, according to an illustrative embodiment of the invention.

[0048] Figure 15 is a block diagram featuring a method of blending paint strokes into a texture layer where the method protects one or more selected regions of the image from subsequent editing, according to an illustrative embodiment of the invention.

[0049] Figure 16 is a block diagram featuring a method of blending erase strokes into a texture layer, where the method protects one or more selected regions of the image from subsequent editing, according to an illustrative embodiment of the invention.

[0050] Figure 17 is a block diagram featuring a method of texture painting in which texture space is dynamically allocated following object modification by the user, according to an illustrative embodiment of the invention.

[0051] Figure 18 is a schematic diagram illustrating a method of texture painting in which a brush stroke falloff is represented in texture space by computing the distance between a point in object space corresponding to a pixel and a point along the center of the brush stroke, according to an illustrative embodiment of the invention.

[0052] Figure 19 is a block diagram featuring a method of texture painting with a brush stroke falloff as illustrated in Figure 18, according to an illustrative embodiment of the invention.

[0053] Figure 20 is a block diagram featuring a computer system architecture for graphically rendering a virtual object, according to an illustrative embodiment of the invention.

[0054] Figure 21 is a block diagram featuring a computer system architecture for graphically rendering a virtual object, according to an illustrative embodiment of the invention.

[0055] Figure 22 is an image of a virtual object whose surface has been painted using a vertex painting method.

[0056] Figure 23 is an image of the virtual object of Figure 22, wherein the surface of the virtual object has been painted by a user in object space using a texture mapping method, according to an illustrative embodiment of the invention.

[0057] Figure 24 is an image of a spherical virtual object with a periodic texture applied along a curved brush stroke, according to an illustrative embodiment of the invention.

[0058] Figure 25 is a schematic diagram illustrating the division of a voxel into six tetrahedra, used in performing a marching tetrahedra algorithm to generate a surface mesh for a virtual object, according to an illustrative embodiment of the invention.

Detailed Description

[0059] The invention provides improved methods of graphically rendering the surface of a three-dimensional virtual object. The invention relates to a method of texture mapping that allows a user to switch between a paint mode and a sculpt mode without losing data. The method also provides for accurate and efficient rendering without requiring global parameterization.

[0060] Texture mapping is a process by which points on the surface of a three-dimensional virtual object are related to points on a two-dimensional surface. Texture mapping allows certain operations to be performed on the three-dimensional object as if it were flat. The three-dimensional virtual object is generally represented in object space, for example, as a system of elements and/or points in a Cartesian (x,y,z) coordinate system. The surface elements of the object are mapped to texture elements in two-dimensional texture space. The texture is then rendered by relating texels, or pixels in texture space, to corresponding points in object space, and displaying them to a user. Additional effects, such as lighting, shadowing, manipulation of orientation and/or size, animation, and other effects, may be performed once the surface of the object is initially rendered.

[0061] Because embodiments of the invention allow a user to paint directly on the object in object space, the texture space need not be coherent. There is no need to display the texture space to the user. Accordingly, mapping methods of the invention do not require global UV parameterization.

[0062] Figure 1A is a schematic diagram 100 illustrating triangular surface elements of a three dimensional virtual object in object space. Figure 1B shows a flat, coherent, two-dimensional texture map 120 of the object of Figure 1A, produced using global UV parameterization. The elements of the texture map 120 are shown in texture space in Figure 1B, and are linked to the surface elements of the virtual object in object space, shown in Figure 1A. For many complex virtual objects, producing a coherent two-dimensional texture with minimal distortion when mapped to the three-dimensional object is a computationally difficult (or intractable) problem that may require manual input and/or user trial-and-error.

[0063] Figure 2A is a schematic diagram 200 illustrating a three-dimensional virtual object as viewed in object space. The virtual object is a three-dimensional block with one face painted with a two-dimensional image of a cell phone. Instead of generating a global UV parameterization, surface elements of the virtual object are mapped to texture elements in a two-dimensional texture space, as shown in the schematic diagram 220 of Figure 2B. Methods of the invention, discussed in more detail herein, are used to fill up texture elements allocated to surface elements of the virtual object as a user paints a virtual object in object space. The texture need not be coherent, since the user does not need to view the texture space when painting.

[0064] Texture mapping includes texture allocation and texture rendering. Texture allocation refers to the process of allocating elements in texture space to corresponding elements in object space.

[0065] Figure 3 is a block diagram 300 featuring a method of texture allocation, according to an illustrative embodiment of the invention. The allocation process of Figure 3 begins when the user enters a paint command, or otherwise indicates she is ready to begin painting or otherwise editing the surface of the virtual object. At this stage, it is possible that no elements in texture space have been allocated to the surface elements of the object. Alternatively, the method in the block diagram 300 of Figure 3 may be initiated when the user enters a paint command after having sculpted the object, added "clay" to the object, or otherwise modified the shape of the object in a way that creates additional surface. In this case, some texture elements have been allocated to the surface elements of the object, but there may be new surface elements of the object requiring the allocation of texture elements in texture space. If jacks are eliminated as a result of the object edits, texture space is deallocated by freeing the texture elements corresponding to the eliminated jacks.

[0066] The method of Figure 3 proceeds by stepping through jacks in the object space in the order in which the object is rendered. Texture rendering, and rendering order, is discussed in more detail hereinbelow. Texture allocation proceeds from jack to jack in render order by allocating texture elements for the surface elements in the jack.

[0067] Texture allocation proceeds such that a plurality of texture regions are defined for the virtual object being rendered. The organization of allocated texture into texture regions is important in the rendering methods described herein, since it allows binding and rendering texture from one texture region at the time, providing faster, more efficient rendering of complex virtual objects. Figure 4A illustrates the partitioning of texture elements -- for example, references 402, 403, and 404 -- associated with a three-dimensional virtual object 400 into texture regions. Although later figures portray texture regions as boxes that divide up the virtual object into neat segments, as seen at reference 406, a texture region is not necessarily so neatly partitioned, since its extent is based on how texture is allocated to the jacks in render order. For example, texture regions may divide up the virtual object roughly into the portions indicated at references 408, 410, 412, 414, and 416.

[0068] Furthermore, jacks may be organized into jack blocks. For example, a jack block may be an 8x8x8 contiguous block of voxels, although any other number or configuration of voxels may be used. The voxels within a given jack block may or may not include surface-containing voxels (jacks). Figure 4B illustrates a volume 420 in object space encompassing the jack blocks that contain all the object surface elements that are mapped to a given texture region. One of the jack blocks is shown at reference 422. The surface of the virtual object that intersects the volume 420 is shown at reference 423. The jack block at reference 422 contains 8x8x8 (= 512) voxels. A single jack is shown at reference 424. Jack 424 contains three object surface elements 426, 428, 430.

[0069] The allocation method produces a list, for example, the "Texture Mapper" depicted in the system architecture of Figure 20 and Figure 21. The list provides a "map" identifying texture elements corresponding to the jacks of the virtual object. The list may be a hash map identifying each texture element corresponding to surfaces in each of a plurality of jacks. More preferably, the list is a hash map of hash maps identifying each texture element corresponding to surface(s) in each jack of each jack block. The Texture Mapper stores all the texture element identifiers owned by a particular jack. For example, Figure 5 illustrates a hash map of hash maps 500 identifying each texture element ("Q0, Q1, Q2, Q3"), within a given texture region ("TR 5"), where each texture element is mapped to a jack ("Local Jack") within a jack block ("Block").

[0070] The texture allocation method of Figure 3 begins by considering the first jack of the first jack block of the virtual object, in render order, and determining in step 302 whether texture has been allocated for this jack. If not, step 304 directs that a first texture region be created if none currently exist, or, alternatively, proceeds to the current texture region. Step 306 determines whether there is enough space in the current texture region to contain the surface elements in the current jack. The computation in step 306 may involve requiring a margin of extra texture elements in each texture region to accommodate possible surface expansion in a jack block, due to edits that change the amount of surface of the virtual object in a given jack block.

[0071] If there is enough space in the current texture region for the current jack, step 310 of Figure 3 directs that a sufficient number of texture elements needed for the current jack be allocated. In an embodiment where texture elements are quadrilaterals (quads) and surface elements are triangles, step 310 may involve pairing surface elements that share a common edge, and mapping each pair to an available quad. If there is not enough space in the current texture region for the current jack, step 308 of Figure 3 directs that a new texture region be created, and that all texture elements corresponding to jacks within the current jack block be moved into the new texture region. Thus, in this instance, the entire jack block's texture elements are moved as a unit to a new texture region where more texture elements are available. Then, entries for the transferred texture elements are updated in the Texture Mapper, accordingly.

[0072] Step 312 of Figure 3 proceeds if texture has previously been allocated for the current jack, or, alternatively, following the allocation of texture for the current jack in step 310. Step 312 determines whether the current jack is the last jack in the current jack block. If not, step 318 proceeds to the next jack in the current jack block, and the allocation process starts again at step 302 with the new jack. If the current jack is the last jack in the current jack block, step 314 determines if there is a jack block remaining. If no jack block remains, the allocation process is complete. If there is a jack block remaining, step 316 proceeds to the first jack of the next jack block, and the process starts again at step 302 with the new jack.

[0073] Texture allocation methods of the invention support dynamic texture allocation and deallocation – that is, the creation and/or elimination of texture elements corresponding to edited portions of the object, without affecting texture previously allocated for texture elements corresponding to unedited portions of the object. In this way, the invention supports a quick transition from a sculpting mode, or other object editing mode, to a painting mode, or other surface editing mode, and vice versa. The allocation method of Figure 3 allocates new texture elements when new jacks are created as a result of object edits. The allocation method deallocates (frees, erases) texture elements when their associated jacks are destroyed. When the surface within a voxel changes, for example, due to surface-based editing such as tugging or any operation involving rasterization, a slightly modified method is performed. First, texture previously allocated to the edited jack is preserved by caching it, then the old texture is deallocated while the new texture is reallocated according to the method of Figure 3. Finally, the cached texture is resampled into the new texture where the surface elements have not changed significantly.

[0074] In addition to texture allocation, texture mapping involves the process of texture rendering. While texture allocation need only be performed if texture has not yet been

allocated for a given virtual object or when there has been a change that affects the amount of surface of the virtual object, texture rendering is performed once for every frame that is graphically rendered. For example, texture rendering may take place up to about 30 times per second, or more, depending on the complexity of the virtual object being rendered and the capability of the computer system and graphics application used. Texture rendering involves sending data representing the contents of each texture element to a graphics rendering interface, such as a graphics card and/or graphics application. Thus, the texture rendering methods of the invention may optionally include using a commercially-available platform for graphical rendering and hardware acceleration. Moreover, the texture rendering methods of the invention may optionally include the use of a commercially-available graphics application programming interface.
Along with sending data representing the contents of each texture element, texture rendering methods of the invention send texture coordinates to the graphics rendering interface/application in order to correctly relate the texture data to corresponding portions of the surface of the virtual object.

[0075] Figure 6 is a schematic diagram 600 illustrating the binding of a blended composite texture 602 for a given texture region to a graphics application and sending texture coordinates to the graphics application via a graphics card 604. Hardware limitations in the rendering of complex virtual objects may be overcome by dividing all the texture that is mapped to a virtual object into a set of texture regions according to the method of Figure 3.

[0076] Figure 7 is a block diagram 700 featuring a method of texture rendering, according to an illustrative embodiment of the invention. Step 702 indicates binding a blended composite texture representing the current texture region to the graphics application. Methods of blending collapsible texture layers to form blended composite textures are disclosed in more detail hereinbelow. In a preferred embodiment, the blended composite texture is a contiguous texture containing all the texture information required by the graphics application in the graphical rendering of the surface of the virtual object. Step 704 indicates the method begins with the first jack of the first jack block of the current texture region; the steps which follow show the order in which the other jacks are considered. In step 706, the rendering method determines two-dimensional texture space coordinates for surface elements in the current jack block, and transmits the information for use by the graphics application. The determination of texture space coordinates for the surface elements in a given jack is an important step in the texture rendering process, and is described in more detail in Figures 8A, 8B, 9, and 10.

[0077] Steps 708, 710, 712, and 714 of Figure 7 show the order in which texture space coordinates are determined for elements within a given texture region. Step 706 is performed for each jack of the current jack block, then for each jack of the next jack block within the current texture region. The process continues until step 712 determines that the last jack of the last jack block in the current texture region has been reached. Then, step 716 determines whether the current texture region is the last region allocated for the virtual object. If so, the rendering process has completed its cycle. If not, step 718 indicates that the process returns to step 702 with the blended composite texture for the next texture region. The method proceeds until all jacks have been rendered.

[0078] In another embodiment, the method of texture rendering proceeds jack block by jack block, irrespective of texture regions. Here, unlike the method shown in Figure 7, jack blocks are not rendered in an order according to the texture region in which they lie. As rendering proceeds from jack block to jack block, the texture region in which the current jack block lies is first bound to the graphics application before texture space coordinates are determined for surface elements in each jack of the current jack block. A single texture region may be bound multiple times, depending on the jack block render order. The render order may be adjusted to reduce the number of texture region bindings required.

[0079] Figures 8A, 8B, 9, and 10 illustrate methods of determining texture space coordinates for the surface elements in a given jack. Figure 8A is a schematic diagram illustrating the use of two lookup tables derived from an index corresponding to the given jack. Each jack has an index that is produced as a result of a mesh generation process for the voxel-based virtual object, indicating which of a known set of surface element configurations describes the current jack. The index may be, for example, a marching cubes index, derived from an implementation of a marching cubes algorithm that is used to generate a surface mesh for the virtual object. Alternatively, the index may be, for example, a marching tetrahedra index, derived from an implementation of a marching tetrahedra algorithm that is used to generate a surface mesh for the virtual object. At a given point in the rendering process, the surface mesh, with its corresponding indices, may be an initial mesh generated for the virtual object, or the surface mesh may be a re-mesh generated due to a change in the surface of the virtual object, for example, following a carving or other object editing function performed by the user.

[0080] Figures 8A and 8B feature a texture rendering method using two lookup tables to determine texture coordinates for each surface element within a given jack. Other embodiments of the invention use one lookup table, or three or more lookup tables. A lookup table may be a list, a file, or may assume any other form that provides information about surface elements of the virtual object using indices from mesh generation. In a preferred embodiment, the lookup tables are pre-computed, prior to use during graphical rendering. Steps 1-4 illustrated in the schematic diagram of Figure 8A correspond to the similarly numbered steps in the block diagram 850 of Figure 8B.

[0081] The method illustrated in Figures 8A and 8B proceeds by using the Texture Mapper to determine the identification of all texture elements that are mapped to surface elements within a given jack. This is performed for each jack of the virtual model, according to the method illustrated in Figure 7, and as indicated at reference 852 of Figure 8B. For example, the jack 802 in Figure 8A contains three surface elements -- labeled Tri 1, Tri 2, and Tri 3 -- that have been allocated two texture elements (here, quadrilaterals, or "quads"), labeled "17" and "201" in the current texture region. The texture elements were allocated according to the texture allocation method illustrated in Figure 3.

[0082] The rendering method of Figure 8B proceeds by performing steps 2, 3, and 4 for each surface element within the current jack, as indicated by reference 854. Step 2 uses the first lookup table to determine in which texture element the current surface element is mapped; step 3 uses the second lookup table to determine to which position within the texture element the current surface element is mapped; and step 4 determines texture coordinates within the current blended composite texture to which the current surface element is mapped.

[0083] Figure 8A illustrates steps 2, 3, and 4, described above, performed for a surface element, labeled Tri 3, in a given jack 802 produced by a marching cubes mesh generation algorithm and having marching cubes index "45". Step 2 of Figure 8A shows that the first lookup table, LUT 1, is used to determine in which of the two quads for this jack the surface element, Tri 3, is mapped. Tri 3 is located in Quad 2 of 2 allocated for jack 802. Step 3 of Figure 8A shows that the second lookup table, LUT 2, is used to determine to which of two possible positions within the texture element the surface element is mapped. Tri 3 is located in the bottom half of Quad 2 of 2 (position 1). Finally, step 4 of Figure 8A shows that global texture coordinates are determined for the surface element, Tri 3, by converting the local texture coordinates of its vertices -- (0,0), (1,0), and (0,1) -- into global texture coordinates within the current texture region using the identification of the texture element, Quad 201, in the Texture Mapper.

[0084] Figure 9 and Figure 10 illustrate methods of creating lookup tables used in the texture rendering method of Figures 8A and 8B. Figure 9 is a block diagram 900 featuring a method of creating a first lookup table indicating in which texture element a given surface element is mapped. The method attempts to pair triangles that share a common edge together into the same quad. For the case of a mesh generated using a marching cubes algorithm, there are at most 4 connected series of triangles in any given jack, where a "series" can be one triangle unconnected to another triangle in the jack, or a "series" can be more than one triangle connected to each other in the jack. Therefore, a jack in this instance can consume at most 4 texture quads, each containing either one or two triangles. The method in Figure 9 of creating the first lookup table finds unpaired triangles with the fewest neighbors 908 by stepping through nested categories: each jack with a known marching cubes index 902, each connected series of triangles 904, and each remaining triangle of a connected series to be paired 906. Step 910 determines whether a given triangle (surface element) has an unpaired neighbor. If so, step 912 pairs the current triangle with its neighbor into an unoccupied quad (texture element). If not, step 914 places the current triangle into an unoccupied quad alone.

[0085] The results of the method of Figure 9 can be seen in Figure 8A. Since Tri 1 and Tri 2 share a common edge, they are paired into a single quad -- Quad 1 of 2 in Figure 8A, which is Quad 17 of the current texture region. Tri 3 shares a common edge with Tri 1, but since Tri 1 is already paired with Tri 2, Tri 3 is allocated its own texture element – Quad 2 of 2 in Figure 8A, which is Quad 201 of the current texture region.

[0086] Figure 10 illustrates the creation of the second lookup table. In this embodiment, when a triangle pair is assigned to an unoccupied quad, the first triangle is assigned to the bottom-left half of the quad, with vertex coordinates at the top-left, bottom-left, and bottom-right corners, while the second triangle is assigned to the top-right half of the quad with vertex coordinates at the top-left, top-right, and bottom-right corners. When a single triangle is placed in an unoccupied quad alone, it is assigned to the bottom-left half of the quad. Thus, Tri 3, shown in Figures 8A and Figure 10, is placed into the bottom-left half of Quad 201. Thus, as seen in Figures 8A and at reference 1004 of Figure 10, Tri 3 has local vertex coordinates (0,0), (1,0), and (0,1). Figure 10 shows that triangle Tri 1 has local vertex coordinates (0,0), (1,0), and (0,1) in Quad 17 and its pair, Tri 2, has local vertex coordinates (1,1), (1,0), and (0,1).

[0087] More sophisticated surface-element-to-texture-element mapping schemes are possible. For example, in one embodiment, where there are a finite number of surface element configurations within a jack that are created during mesh generation, texture elements indicative of each possible surface element configuration within a jack can be pre-computed so that a single texture element is mapped to <u>all</u> the surfaces within a given jack.

[0088] When the user paints a virtual object in object space, values are assigned to corresponding texels in texture space. The user "paints" texture elements, in essence, via the link between surface elements of the virtual object in object space and corresponding texture elements in texture space. This process is invisible to the user, who only sees the effects of the painting operation in object space. However, the process of blending brush strokes, occurs in texture space using two-dimensional texture compositing techniques. Graphical user input fills texture layers corresponding to a given texture region; then, the texture layers are combined into a blended composite texture for that texture region. The blended composite texture for each texture region is bound one-at-a-time to a graphics application during texture rendering, as described in more detail elsewhere herein.

[0089] Figure 11 features a schematic diagram 1100 illustrating the blending of texture layers 1102, 1104, 1106, 1108, 1110 to form a composite blended texture 602 for a given texture region. Blending may include performing one or more texture compositing operations. There may be any number of texture layers that are blended to form a given blended texture 602. The texture layers may include, for example, scratch textures, paint textures, and/or stencil textures.

[0090] A scratch texture indicates one or more intensity values of pixels (and/or texels) associated with each brush stroke. A scratch texture may be thought of as a "template" describing the brush characteristics; for example, the scratch texture may contain intensity values that reflect brush size and/or edge effects. The brush may be assigned a certain width, a varying width along the length of a stroke, a certain shape characteristic of the end of the brush, and/or a "falloff" or transition region along a portion of the brush, such as the edges of the brush. The intensity values in the scratch texture may be used to determine the intensity of a selected color at a given pixel within the brush stroke. For example, a scratch texture comprising values representing a given brush stroke may represent a "falloff" region of the brush stroke by assigning intensity values from 1 (full intensity) at the center, or near to the center, of the stroke, down to 0 (no intensity) at points at the edges of the stroke. Thus, a blended composite of the scratch texture with a paint texture containing a single monochromatic color value would portray a paint stroke with the most intense color at the center of the stroke, falling off to no color at the very edges of the stroke.

[0091] As used herein, a brush stroke is an operation performed on a collection of pixels in a texture and/or image. Brush strokes include, for example, paint strokes, erase strokes, pen strokes, lines, characters, and text. The manner in which a brush stroke operates on a collection of pixels may be determined interactively by a user, as in the performance of a paint or erase stroke, using a graphical interface device, or non-interactively – for example, in batch – without any user input. Thus, brush strokes include, for example, batch deletions, batch pastes, and flood fills. The pixels/texels on which a brush stroke operates may be either contiguous or non-contiguous.

[0092] The texture layers 1102, 1104, 1106, 1108, 1110 portrayed in Figure 11 may also include stencil textures. Stenciling allows a user to more accurately control the editing of an image by specifying a maximum amount by which selected portions of the image are allowed to change over the course of a series of brush strokes. Stencil textures are described in more detail herein, for example, in the descriptions of Figure 15 and Figure 16.

[0093] Texture layers may be stored in "collapsed" form, as shown in Figure 12, thereby requiring less memory and improving the efficiency of the rendering process. As shown in Figure 11, the collapsed textures are combined to form a contiguous blended texture suitable for binding to a graphics application. Alternatively, the blended texture may be stored in collapsed form.

[0094] A representative contiguous texture 1200 is shown in Figure 12. The contiguous texture 1200 contains quadrilateral texture elements. Other shapes may be used, alternatively. All texels of each of the elements of the contiguous texture 1200 are represented in normal "expanded" form. Where a given texture element contains limited data, for example, a single color value, it is not necessary to store this value for each of the texels of the given element. Thus, a collapsible texture 1202 can save memory by representing uniform texture elements with the limited data, for example, the single color value applied to all texels of the element.

[0095] The collapsible texture 1202 in Figure 12 is a grid of elements indicating uniform texture elements, non-uniform texture elements, and locations of the nearest free texture element in the grid. Here, an element of the grid may be a value of any bit length (i.e. 32-bit) that represents, for example, the color value of a uniform (collapsed) texture element 1208, the location of the next free grid element 1210, or the location 1212 of a non-uniform (expanded) texture element 1206, as shown in the legend 1204 of Figure 12. For each collapsible texture 1202, there may be a shadow texture that is used to interpret what kind of information the values of the grid 1202 represent. For example, a shadow texture may comprise codes corresponding to the elements of the grid 1202 that indicate whether a grid element is a uniform quad value, a location or identification number of a free quad, or a pointer to a non-uniform quad.

[0096] Texture layers are combined in discrete time slices to form the blended composite image in a pixel-by pixel manner, without interrupting application of additional brush strokes by the user. Each brush stroke within a given texture region may be represented using a scratch texture. An individual brush stroke is divided into a series of stroke segments that are separately blended into the scratch texture. The invention provides a method that prevents double-blending of overlapping portions of segments of the individual brush stroke, thereby avoiding undesired artifacts at the segment ends.

[0097] Figure 13 shows a schematic diagram 1300 illustrating two segments of a brush stroke both with 1302 and without 1304 a double-blending artifact in the overlapping portion. Each segment of the brush stroke is represented with a pillbox shape. Brush stroke 1302 is divided into two segments 1306 and 1308, and brush stroke 1304 is divided into two segments 1310 and 1312. The overlapping portion of the two segments in brush stroke 1302 results in higher intensities in the circular region 1314. The blending method performed for brush stroke 1304 avoids the higher intensities in the circular region where the ends of the pillboxes overlap, resulting in a brush stroke of even intensity.

[0098] The blending method uses a scratch texture to describe the size and shape of the brush. By assigning a new pixel value to the scratch texture only if it exceeds the existing value at that pixel, the double-blending artifact is avoided. The blending method includes the steps of: (1) receiving brush stroke data from a graphical user interface; (2) for each of a plurality of pixels of a scratch texture, comparing a received pixel value to a value previously written at the corresponding pixel of the scratch texture and assigning the received value to the corresponding pixel of the scratch texture only if it exceeds the existing value; and (3) blending the scratch texture into the target image. Here, the scratch texture is a texture having one or more intensity values of pixels (and/or texels) associated with each brush stroke. The blending step may be performed substantially upon completion of the performance of the paint stroke by the user. Each scratch texture is emptied or deleted after its contents are blended, so that each new brush stroke fills an empty scratch texture.

[0099] In order to enhance interactivity, texture layers are blended in the order in which the brush strokes they represent are performed. Texture data is blended pixel-by-pixel in discrete time slices, without interrupting the user's application of additional brush strokes. Thus, multiple scratch textures layers may be blended together where each scratch texture represents an individual brush strokes within a given texture region. This technique is demonstrated in Figures 14A and 14B.

[0100] Figure 14A is a schematic diagram 406 conceptually illustrating the partitioning of a three dimensional object in object space into multiple texture regions. Figure 14B illustrates the blending of multiple texture layers for each texture region in which a brush stroke falls. When the user begins a brush stroke, a new scratch layer is added for the texture region into the topmost layer. For example, Figure 14B shows brush stroke 1401 applied by a user in object space, conceptually depicted as affecting texture in texture regions 1402, 1404, and 1406; brush stroke 1405 affects texture regions 1406, 1408, and 1412; and brush stroke 1409 affects texture regions 1410, 1406, and 1412. As the new scratch layer is added for a given texture region into the topmost layer, the bottom-most scratch layer for that region is being blended into the blended composite texture for that region. The raised tiles 1402, 1404, 1406, 1408, 1410, and 1412 in Figure 14B represent the topmost layer for their respective texture regions.

[0101] The texture layers that are combined to form a blended composite texture for a given texture region may include stencil textures. Stencil textures are used to protect a selected region of an image from subsequent editing. The methods of blending texture are compatible with stenciling methods that allow a user to more accurately control the editing of an image by specifying a maximum amount by which selected portions of the image are allowed to change over the course of a series of brush strokes.

[0102] The stenciling methods involve protecting an image using a first texture and a second texture, rather than a single texture alone. Figure 15 is a block diagram 1500 featuring a method of blending brush strokes into a texture layer protected by application of a stencil. The first texture 1502 includes texels with values indicating levels of protection to be applied to corresponding texels of the protected texture 1504. For example, the first texture 1502 may be a map of values indicating one or more regions of the texture to be protected from editing by subsequent paint strokes. The level of protection may be from 0% to 100%, where 0% indicates no protection from subsequent edits, and 100% indicates full protection from subsequent edits. A level between 0% and

100% indicates partial protection, and may correspond to a minimum opacity above which to maintain at least a portion of the protected region throughout the application of subsequent brush strokes.

[0103] Overlapping portions may result from multiple overlapping brush strokes and/or from a single brush stroke that overlaps itself. Despite the presence of any overlapping portion(s), and despite the number of brush strokes applied following activation of the stencil, methods of the invention can prevent the opacity of an initial version, or initial layer, of the selected region(s) from decreasing below a specified minimum in any subsequent composite.

[0104] The stenciling method illustrated in Figure 15 proceeds by directing graphical input, for example, brush strokes 1506, into a second texture 1508, rather than directly blending the graphical input into the protected texture 1504. The second texture 1508 acts as a buffer accumulating graphical input. The protected texture 1504 may be combined along with other texture layers into a blended composite texture for the given texture region, or, alternatively, the protected texture 1504 may, in fact, be the blended composite texture for a given texture region.

[0105] The interactivity of the user's image editing experience may be preserved by use of a display texture 1510. In one embodiment, texels of the protected texture 1504 are copied directly into a display texture 1510, while texels of the second texture 1508 are modified according to the first texture 1502, then blended into the display texture 1510.

[0106] User interactivity is preserved by modifying texels of the second texture and blending the modified texels into the display image on a texel-by-texel basis. In this way, the user sees the resulting image emerge, subject to the user-specified protection, as the user continues to apply brush strokes.

[0107] The display texture can reflect real-time user brush strokes, as well as preserve a minimum opacity of the original texture layer within a protected region, regardless of the number of brush strokes that follow. This is because, in a preferred embodiment, each update of the display texture is performed by: (1) re-copying texel values of the original protected texture layer into the display texture texels, and then (2) compositing the modified second texture texels with the display texture texels. The display texture 1510 may be updated at a rate of up to about 30 times per second, or more, depending on the complexity of the image and the computational speed for graphical rendering. The update rate may be less for more complex images or for slower machines – for example, from about 10 times per second to about 20 times per second.

[0108] The use of a display texture 1510 is optional. Whether or not a display texture 1510 is used, methods of the invention can preserve a minimum opacity of the original image layer within a protected region by accumulating graphical input in the second texture 1508, modifying the second texture 1508 using the first texture 1502, and, subsequently, blending the modified second texture 1508 into the protected texture 1504.

[0109] The user may provide one or more signals indicating the beginning and/or ending of the period of time in which brush strokes are accumulated in the second texture. The user provides a first signal, such as a button click, to indicate the beginning of the application of a stencil. Alternatively, the stencil may self-activate once the user defines the first texture. The step of defining the first texture may include indicating: (1) one or more regions of the surface of a virtual object to be protected; and/or (2) one or more levels of protection to apply within the one or more regions. The surface elements within the region indicated by the user in object space are mapped to corresponding texture elements in texture space.

[0110] Once the stencil is active, graphical input representing brush strokes by the user are directed into the second texture 1508. When the user has completed one or more brush strokes, the user may provide a second signal to deactivate the stencil. In one embodiment, once the stencil is deactivated, the second texture 1508, modified by the first texture, is blended into the protected texture 1504.

[0111] Real-time display of erase strokes requires modification of paint strokes applied both before and after activation of the stencil. Figure 16 is a block diagram 1600 featuring a stenciling method in which erase strokes 1602 are blended into a protected texture 1504. Generally, the second texture 1508 contains data from paint strokes applied after activation of the stencil, but not before. Therefore, the step of modifying a value of one or more texels of the protected texture 1504 is performed *prior to* copying texels of the protected texture 1504 into the display texture 1510 and compositing the modified second texture texels with the display texture texels. Texel values of the protected texture 1504 are modified according to the level(s) of protection indicated in the first texture 1502.

[0112] In one embodiment, a minimum opacity of an original texture layer may be preserved despite repeated erase strokes within the protected region, where overlapping erase strokes result in the attenuation of a texel value of the protected texture down to a minimum value, but no further. The minimum value is determined from the protection level of that texel as indicated in the first texture 1502. In cases where texels are represented by RGBA quadruples, the first texture 1502 may be used to derive a minimum alpha channel value below which the texel values of the protected texture 1504 are not allowed to fall. Pixel values and/or texel values include values of any bit depth, where bit depth generally ranges from 1 to 64 bits per pixel. Pixel values include grayspace values, RGB colorspace values, RGBA colorspace values, or any other colorspace values, such as HSL, HSV, CMY, CMYK, CIE Lab, and R-Y/B-Y. Preferred methods of the invention are performed using 24-bit, RGBA colorspace pixels, although any bit depth and/or colorspace format may be used.

[0113] The display methods discussed herein work where a user performs paint strokes, erase strokes, or both paint and erase strokes, while the stencil is active. Additionally, these display methods preserve the interactivity experienced by the user during the editing process, while maintaining the protection offered by the stenciling methods described herein.

[0114] Methods of the invention involve the modification of paint colors of images and/or textures, as well as the blending of paint colors between two or more images and/or textures. For example, in one embodiment, pixels of the second texture 1508 are modified using the first texture 1502, where the first texture 1502 is a map of values indicating protection levels for the protected texture 1504 and the second texture 1508 contains accumulated graphical input. A texel of the second texture 1508 may be modified by attenuating its color value according to the protection level in the first texture 1502 corresponding to that texel.

[0115] Blending of images and/or textures may be performed using one or more compositing operations. For example, in one embodiment, texels of the second texture

1508 are blended into the protected texture 1504. This may be done by performing a compositing operation, such as an "overlay" operation in RGBA colorspace, between the second texture 1508 and the protected texture 1504.

[0116] Because texture is dynamically allocated, a user may seamlessly switch between an object modification mode (for example, a sculpt mode) and a surface paint mode without losing paint information for unaffected surfaces of the virtual object. Figure 17 is a block diagram 1700 featuring a method of texture painting in which texture space is dynamically allocated following modification of the virtual object. Figure 17 may represent part or all of a graphics thread for graphically rendering the virtual object. Throughout the process depicted in Figure 17, the user may apply any number of brush strokes directly onto the three dimensional object in object space using, for example, a haptic interface device. The user may also switch from the painting feature to a sculpting feature, and back again. For example, the method of Figure 17 may be initiated every time the user enters the painting feature, for example, by clicking an icon or button, or by providing some other signal.

[0117] Step 1702 of the painting method illustrated in Figure 17 allocates texture space for the virtual object according to methods described herein, for example, the texture allocation method depicted in Figure 3. Step 1704 describes the creation and rendering of blended composite textures for each of a plurality of texture regions, as shown, for example, in Figures 7 and 11. Step 1704 may be performed multiple times per second, for example, up to about 40 times per second or more. For more complex virtual objects, step 1704 is performed at a slower rate – for example, from about 10 Hz to about 30 Hz.

[0118] Step 1704 of Figure 17 performs a set of substeps for each of a plurality of texture regions, as indicated at reference 1706. The substeps include updating texture layers according to user brush strokes 1708, combining the texture layers into blended composite textures 1710, binding the composite texture to a graphics application (where the graphics application provides hardware and/or software for display of the virtual object in object space) 1712, and transmitting texture coordinates to the graphics application 1714. After one or more render cycles of step 1704, the method determines in step 1716 whether the user has modified the virtual object such that any surface is destroyed or new surface is created. If not, the creation and rendering of blended composite textures of step 1704 continues. If surface is created or destroyed, the surface mesh for the virtual object is updated according to the user modification. For example, if the user has sculpted the object and has re-entered the painting feature, step 1718 updates the surface mesh by remeshing at least a portion of the virtual object. The mesh generation step may employ a marching cubes algorithm, for example, if the model is a voxel-based model. Then, the method of Figure 17 proceeds to step 1702, where texture is allocated, for example, according to the method of Figure 3, which prevents reallocation of texture for surface elements in unmodified jacks. The creation and rendering of blended textures in step 1704 then continues as before.

[0119] An embodiment of the painting method of Figure 17 allows a user to paint with brush strokes whose characteristics are reflected in corresponding scratch textures. For example, each brush stroke may include a falloff region, or transition region, near the edges of the brush stroke such that the paint appears to fade into the background color at the edges of the brush. Each stroke is divided into segments that link positions of a graphical interface device, such as a stylus or other painting tool. The positions are recorded at successive frames as the user moves the graphical interface device in real space, and, preferably, while the user views a graphical rendering of the virtual object in object space. The display may represent the virtual object using a Cartesian coordinate system, or any other coordinate system. Surface elements of the object which fall within a volume of influence of the tool are painted.

[0120] Figure 18 is a schematic diagram 1800 illustrating a method of painting a surface element 1802 affected by a brush stroke 1801, where the brush stroke 1801 has a falloff region at its edges. Similar methods are performed for brush strokes with other attributes or characteristics. Triangle 1802 represents a surface element in object space, and triangle 1802' shows the surface element with the path of a user's brush stroke 1801 superimposed. Since triangle 1802 falls within a volume of influence of the brush stroke 1801, it will be painted. The surface element 1802 is mapped to quadrilateral 1804 in texture space, according to the texture mapping methods discussed herein. Color intensities corresponding to the brush stroke 1801 are assigned to texels of the texture element 1804, which is used to graphically render the surface element 1802 in object space. Since the paint stroke has a falloff region at its edges, there may be a variation of color intensities assigned to the texture element 1804. Figure 18 shows how an intensity is assigned to a texel of the texture element 1804 following its scan conversion based on the position of vertices v1, v2, and v3 of the corresponding surface element 1802.

[0121] Figure 19 shows a block diagram 1900 of representative steps of the method illustrated in Figure 18. Reference 1902 indicates that the steps which follow are performed for each surface element within the volume of influence of the brush stroke.

An example surface element 1802 within a volume of influence of a brush stroke 1801 is depicted in Figure 18. Step 1904 of Figure 19 finds vertex coordinates in texture space of the texture element corresponding to the surface element in object space. For the surface element 1802 in Figure 18, the vertex coordinates v1, v2, and v3 of the texture element 1804 are determined in step 1904. Then, reference 1906 of Figure 19 indicates that steps 1908, 1910, and 1912 which follow are performed for each texel of the current texture element. Step 1908 computes, for the current texel of the current texture element, a corresponding point vi in object space by interpolating with respect to the vertex positions in texture space. For example, in Figure 18, for the texel centered at position vi in texture element 1804 in texture space, the point vi is located in triangle 1802' in object space by interpolating with respect to the vertex positions v1, v2, and v3 in texture space. Once the point in object space is located, step 1910 of Figure 19 calculates the nearest point on the stroke skeleton, and/or the shortest distance to the stroke skeleton, in order to determine a "falloff" value. For example, in Figure 18, the point vs on the stroke skeleton nearest point vi is located, and the distance D is computed. The "falloff" value, or attenuation factor, to apply at point vi is then computed as a function of D. Step 1912 of Figure 19 then uses the falloff value to write an attenuated version of paint color into the texel. For example, in Figure 18, the computed falloff value based on distance D is used to attenuate a color value chosen for the paint stroke 1801, and the attenuated color value is written into the texel centered at vi in the texture element 1804.

[0122] Figure 20 is a block diagram featuring an example computer system architecture 2000 for graphically rendering a virtual object according to methods of the invention. The architecture 2000 in Figure 20 is presented in UML format. The legend 2001 describes how connected classes and textures of the architecture are related. The workspace data 2002 contains a class for storing all of the texture data, designated the "Texture Manager" 2004. The "Texture Region" block 2006 in Figure 20 represents a container for all texture classes in each of the texture regions of a virtual object being graphically rendered. For example, a virtual object with three texture regions has three "Texture Region" 2006 containers. The "Texture Mapper" 2008 is a list that identifies texture elements corresponding to each of the jacks of the virtual object being graphically rendered. For example, the "Texture Mapper" 2008 may be a hash map of hash maps identifying each texture element corresponding to surface(s) in each jack of each jack block, as shown in the schematic diagram 500 of Figure 5.

[0123] For each texture region, there is an interface, labeled "IComposite Color Texture Region" 2010 in the architecture 2000 of Figure 20, that represents a stack of texture layers for the texture region. This interface 2010 is implemented by a texture that blends and stores paint layers (and/or brush stroke layers), labeled "Composite Color Texture Region" 2012. This texture 2012 contains a texture storing color per texel, labeled "Color Texture Region Component" 2016 in Figure 20, as well as multiple textures storing paint layer data, labeled "Color Texture Region Layer" 2022. The "Color Texture Region Component" 2016 is a contiguous texture. A contiguous texture may be needed for proper binding of a blended composite texture with a computer graphics application for graphical rendering display.

[0124] The "Texture Region Component" 2014 in the architecture of Figure 20 is a template for a contiguous texture class. The "Quad Texture Region Component" 2018 is a template for a "collapsible" texture class, which can be instantiated as a "Color Quad Texture Region Component" 2020, the texture storing color per texel in collapsed form.

[0125] Figure 21 depicts a computer system architecture 2100 that shares a number of components with the architecture 2000 of Figure 20, but has some additional components. The additional components include a class managing the blending of paint strokes, "Texture Stroke Manager" 2104, a class managing the undoing of paint removal during modifications of the virtual object (i.e. during sculpting of the object), "Texture Undo Manager" 2106, and a class for allocating texture, "Texture Allocator" 2108. The architecture 2100 in Figure 21 also further specifies that the "Composite Color Texture Region" 2012 texture for storing and blending paint layers contains multiple "Scratch Texture Region Component" 2112 textures for storing temporary paint strokes (scratch textures), as well as a "Selection Texture Region Component" 2114 texture for storing a user selection, such as a stenciled region. Textures 2112 and 2114 instantiate the template for collapsed textures -- the "Quad Texture Region Component" 2018 -- via a texture that stores intensity per pixel, called the "Intensity Quad Texture Region Component" 2110.

[0126] Any of the methods described herein for graphically rendering a virtual object may be combined with haptic rendering of the virtual object. For example, an embodiment of the invention haptically renders the virtual object in real time as the user applies a brush stroke onto the surface of the virtual object in object space. The haptic rendering process includes determining a force feedback corresponding to a position of a haptic interface device held by the user as the user applies a brush stroke using the device. The force is delivered to the user via the haptic interface device. A haptic rendering process is described, for example, in U.S. Patent No. 6,552,722, by Shih et al., the text of which is hereby incorporated by reference in its entirety.

[0127] A combined haptic and graphical rendering method of the invention performs haptic rendering at a substantially faster rate than graphical rendering. This may be necessary to provide realistic force feedback to the user. In one embodiment, graphical rendering is performed within a range from about 5 Hz to about 150 Hz, while haptic rendering is performed within a range from about 700 Hz to about 1500 Hz. In another embodiment, haptic rendering is performed at about 1000Hz, while graphical rendering is performed at up to about 40 Hz. Haptic and graphical rendering of a virtual object may be performed by different threads of an overall rendering process.

[0128] Figures 22 and 23 demonstrate an improvement obtained using graphical rendering methods of the invention. Both Figures 22 and 23 feature the same virtual object whose surface has been painted by a user in object space. Figure 22 is an image 2200 of a virtual object whose surface has been painted using a vertex painting method. A vertex painting method assigns color values to vertices of surface elements of the object and assigns either averaged or interpolated color values to pixels of the surface elements. There is poor resolution of the painted features on the surface of the virtual object, particularly on relatively flat features of the object such as the face, where surface elements are large.

[0129] Figure 23 shows an image 2300 of the virtual object, where the surface of the virtual object has been painted by a user in object space using the texture mapping methods of the invention. Improved resolution is achieved by employing texture mapping methods of the invention. As seen in Figures 22 and 23, lighting, shadowing, and mirroring effects may be added in the rendering of a virtual object.

[0130] Figure 24 is an image of a spherical virtual object with a periodic texture applied along a curved brush stroke. The periodic checkerboard pattern is used in place of the falloff brush stroke in the texture rendering methods described herein, for example, with regard to Figures 18 and 19. Figure 24 demonstrates the blending of a paint layer with a scratch layer, wherein the paint layer is not simply a monochromatic color. In Figure 24, the blending of the paint layer and scratch layer form a checkerboard pattern about the width of the brush stroke, whose center is indicated by a red line.

[0131] Figure 25 is a schematic diagram illustrating the division of a voxel into six tetrahedra, used in performing a marching tetrahedra algorithm to generate a surface mesh for a virtual object. As discussed elsewhere herein, an index is produced for each surface-containing voxel resulting from a mesh generation algorithm. The index corresponds to one of a finite number of surface element configurations, where the number of possible configurations depends on the mesh generation algorithm employed. In one example, the index is a marching tetrahedra index, derived from an implementation of a marching tetrahedra algorithm used to generate a surface mesh for a virtual object. By splitting up a cube into multiple tetrahedra, triangle vertices may lie along a diagonal of the voxel cube, increasing the resolution of the triangle mesh while keeping the resolution of the voxel grid constant.

[0132] The sampling structure in a marching tetrahedra approach is a tetrahedron. Samples are at the vertices on a rectangular grid, where a mesh cell is made up of eight samples of the grid, and the mesh cell is split into six tetrahedra that tile the cell. Figure

25 shows a mesh cell 2500 that is split into six tetrahedra 2502, 2504, 2506, 2508, 2510, 2512, where vertices of the tetrahedra coincide with vertices of the cube. The configuration of triangular surface elements through each tetrahedron is determined by a lookup table. The index into the lookup table for a given tetrahedron is determined from values at the four vertices of the tetrahedron, where each value indicates whether the vertex is interior or exterior to the isosurface of the virtual object being meshed. The sum of all triangular surface elements for all tetrahedra of a given cube tile the cube. The sum of all triangular surface elements for all the cubes tile the object space.

[0133] A computer hardware apparatus may be used in carrying out any of the methods described herein. The apparatus may include, for example, a general purpose computer, an embedded computer, a laptop or desktop computer, or any other type of computer that is capable of running software, issuing suitable control commands, receiving graphical user input, and recording information. The computer typically includes one or more central processing units for executing the instructions contained in software code that embraces one or more of the methods described herein. The software may include one or more modules recorded on machine-readable media, where the term machine-readable media encompasses software, hardwired logic, firmware, object code, and the like. Additionally, communication buses and I/O ports may be provided to link any or all of the hardware components together and permit communication with other computers and computer networks, including the internet, as desired.

Equivalents

[0134] While the invention has been particularly shown and described with reference to specific preferred embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims.

[0135] What is claimed is: